基于高斯扰动和指数递减策略的改进蝙蝠算法 *

宋一民 [1, 2]，李　煜 [2, 3†]

(1. 郑州财经学院 现代物流与管理系，郑州 450000; 2. 河南大学 商学院，河南 开封 475004; 3. 河南大学 管理科学与工程研究所，河南 开封 475004)

摘　要：针对基本蝙蝠算法后期收敛速度慢、收敛精度不高、稳定性不强等问题，提出基于高斯扰动和指数递减策略的改进蝙蝠算法（GDEDBA）。把指数递减策略引入速度更新公式，使算法迅速进入局部寻优并展开精确搜索；构造高斯扰动项加入到局部新解产生公式，使局部新解中所有粒子与当前全局最优粒子产生信息交流与学习，防止陷入局部最优，增加种群多样性；设计扰动控制因子来控制高斯扰动的扰动范围，增强算法的稳定性。15 个测试函数的仿真结果表明，改进算法的寻优性能显著提高，收敛速度更快，求解精度更高，稳定性更强。

关键词：蝙蝠算法；高斯扰动；指数递减策略；算法改进；函数优化

中图分类号：TP301.6　　doi: 10.19734/j.issn.1001-3695.2018.10.0802

Improved bat algorithm based on Gaussian disturbance and exponential decreasing strategy

Song Yimin[1, 2], Li Yu[2, 3†]

(1. *College of Logistics & Management*, *Zhengzhou Institute of Finance & Economics*, *Zhengzhou* 450000, *China*; 2. *Business School*, *Henan University*, *Kaifeng Henan* 475004, *China*; 3. *Research Institute of Management Science & Engineering*, *Henan University*, *Kaifeng Henan* 475004, *China*)

Abstract: Aiming at the shortcomings of the basic bat algorithm such as slow convergence, low convergence precision and weak stability, this paper designed an improved bat algorithm (GDEDBA) based on Gaussian disturbance and exponential decreasing strategy. It introduced the exponential decreasing strategy into the speed update formula, could enable the algorithm to enter local optimization quickly and exactly; it added the constructed Gaussian disturbance term to the local new solution generation formula, then the information exchange and study between all particles in the local new solution and the current global optimal particles cloud happened, prevented falling into local optimum and increased population diversity; it designed the disturbance control factor to control the disturbance range of Gaussian disturbance, enhanced the stability of the algorithm. The simulation results of 15 classical test functions shows that the optimization performance of improved algorithm is significantly improved, the convergence speed is faster, the solution accuracy is higher, and the stability is stronger.

Key words: bat algorithm; Gaussian disturbance; exponential decreasing; algorithm improvement; function optimization

0　引言

2010 年，剑桥大学学者 Yang 通过模拟蝙蝠回声定位行为，提出一种新的智能优化算法——蝙蝠算法(bat-inspired algorithm)[1]。蝙蝠算法自提出以来，以其模型简单、易于实现等优点被广泛应用于多个领域，如函数优化[2,3]、工程结构设计[4~6]、PID 控制器参数估计[7]、调度问题[8,9]、图像处理[10,11]等。但是蝙蝠算法存在后期收敛速度慢、易陷入局部最优，稳定性差等缺点。为此，研究者对此作了许多改进。Selim 等人[12]用惯性权重因子来控制上一步速度的影响大小，从而来平衡全局和局部搜索之间的强度。 Meng 等人[13]融合栖息地选择和多普勒效应改进蝙蝠算法(NBA)，进一步模仿了蝙蝠的生物学行为对蝙蝠算法改进。Luo 等人[14]提出双子群莱维飞行蝙蝠算法(DLBA)。此外还有混合蝙蝠算法[15~18]。针对蝙蝠算法的不足，本文提出一种新的改进算法(GDEDBA)。

1　基本蝙蝠算法

蝙蝠是一种神奇的物种，是地球上目前为止发现的唯一一种会飞的兽类。其中的微型蝙蝠由于视觉退化，靠回声定位来代替视觉功能。微型蝙蝠在飞行过程中，不断发出 25～150 千赫兹范围内的超声波，超声波遇到障碍物返回到微型蝙蝠的双耳，微型蝙蝠通过判断发出声波到收到返回声波的时间差感知障碍物的距离，通过返回声波的时间间距感知障碍物在哪里，利用回声的响度变化感知障碍物是什么，以此构建三维立体图像。微型蝙蝠利用多普勒效应对周围场景作出判断，离猎物越近，波长越短，频率越高；离猎物越远，波长越大，频率越低，比并且可以判断猎物的移动速度。

蝙蝠算法模拟微型蝙蝠搜索猎物，是一种基于种群的随机寻优算法，问题的解被抽象成搜索空间中的蝙蝠粒子，每只蝙蝠都有一个由优化问题决定的适应度与之对应，蝙蝠个体通过调整频率、响度、脉冲发射率，追随当前的最优蝙蝠在解空间中搜索。蝙蝠粒子在位置 X_i 以速度 V_i 随机飞行，以固定频率 f_{min}（或 λ ）、可变化波长 λ （或 f ）和响度 A_0 搜索猎物，根据猎物与自己的距离调节发射出的脉冲波长（或频率），并在靠近猎物时调整发射脉冲的频度 $r\in[0,1]$，其中 0 表示没有脉冲，表示刚发现猎物，在接近猎物的过程中，逐

收稿日期：2018-10-22；**修回日期：**2018-12-10　**基金项目：**国家自然科学基金资助项目（71601071）；国家教育部人文社科青年基金资助项目（15YJC630079）；河南省重点研发与推广专项资助项目（182102310886）；河南省科技攻关重点项目（162102110109）

作者简介：宋一民（1990-），女，河南林州人，硕士，主要研究方向为智能算法、物流管理；李煜（1969-），女（通信作者），教授，博士，主要研究方向为智能算法、电子商务、物流管理（lyhenu@163.com）.

渐调整为最大脉冲 1。而响度则在搜索猎物过程中从最大值（正值） A_0 到最小值 A_{min} 。

蝙蝠位置 X_i^t 和速度 V_i^t 在步骤 t 时的更新公式如下：

$$f_i = f_{min} + (f_{max} - f_{min})\beta \,, \quad (1)$$

$$V_i^t = V_i^{t-1} + (X_i^{t-1} - X^*) f_i \,, \quad (2)$$

$$X_i^t = X_i^{t-1} + V_i^t \quad (3)$$

其中： $\beta \in [0,1]$ 是一个随机向量； X^* 是当前全局最优位置。

微型蝙蝠具有群体学习能力，能追随当前最接近猎物的蝙蝠个体搜索猎物。受此启发，蝙蝠算法设计局部新解的更新机制，在当前全局最优解的附近产生局部新解，局部新解的更新公式如下：

$$X_{new}^t = X^{*t} + \varepsilon A^t \,, \varepsilon \in [-1,1] \quad (4)$$

其中：$\varepsilon \in [-1,1]$ 是随机数； $A^t = <A_i^t>$ 是所有蝙蝠在这一代里的平均响度。

脉冲发射的响度 A_i 和速率 r_i 也要随着迭代过程进行更新。脉冲发射的响度 A_i 和速率 r_i 的更新公式如下：

$$A_i^t = \alpha A_i^{t-1} \quad (5)$$

$$r_i^t = r_i^0 [1 - e^{-\gamma t}] \quad (6)$$

其中： α 和 γ 是恒量,0< α <1 和 γ >0。

2　基于高斯扰动和指数递减策略的改进蝙蝠算法

2.1　指数递减策略分析

蝙蝠算法和粒子群算法都是通过粒子在解空间中进行全局搜索和局部精确搜索相结合的方法来实现最终寻优的。有效的全局搜索可以帮助粒子跳出局部最优解，有效的局部搜索则帮助粒子在当前搜索区域进行精确搜索，利于算法收敛。那么，如何来平衡全局搜索能力和局部搜索能力，无疑会对算法的寻优性能产生巨大影响。为此，许多专家学者进行了大量研究工作，PSO 算法在此方面的改进策略主要有线性递减权重策略[19]、自适应权重策略[20]、随机权重策略[21]。

文献[22]受线性递减权重策略的启发，构造三种不同策略对粒子群算法改进。实验结果表明，指数递减策略效果最佳。针对基本蝙蝠算法后期收敛速度慢且易在全局最优解附近产生震荡的不足，受文献[22]启发,用指数递减策略对蝙蝠算法进行改进。指数递减策略因子 w 的表达式为

$$w = w_{min} * (w_{max} / w_{min})^{1/(1+ct/t_{max})} \quad (7)$$

其中： w_{max} 、 w_{min} 分别代表 w 的最大值和最小值； t 代表当前迭代次数； t_{max} 代表最大迭代次数；参数 c 的取值选取与文献[22]一致， c =10，取 w_{max} =0.9， w_{min} =0.4。

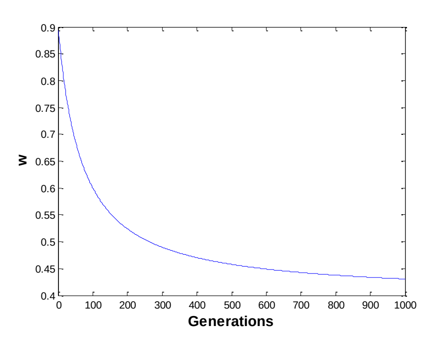


图 1　指数递减策略因子 w 的迭代曲线

Fig. 1　Iterative curve of exponential decreasing strategy factor w

指数递减策略因子随迭代次数的变化如图 1 所示。通过观察： w 在迭代初期随着迭代次数增加以很快的速率递减，而在迭代的中后期， w 下降速度逐渐变得缓慢。在基本蝙蝠算法中， w 是个恒定值 1，容易造成算法在迭代前期陷入局部最优，在迭代后期收敛速度变慢。这里将指数递减策略因子 w 加入到基本蝙蝠算法的速度更新公式中，可以使算法迅速进入局部寻优，并有效展开精确搜索，确保找到最优解。这样一来，既有利于提高算法的收敛速度，又有利于提高算法的寻优精度。

为了测试该策略对算法性能的有效性，对比分析本文的改进算法（GDEDBA）与删除该策略的算法（BA1），在保证相关参数一致的基础上，分别独立运行五个测试函数，取其平均值进行求解精度对比，结果如表 1 所示。给出 GDEDBA 与 BA1 运行函数 F1 的收敛对比图进行收敛速度对比，结果如图 2 所示。

表 1　GDEDBA 与 BA1 的仿真结果

Table 1　Simulation results of GDEDBA &BA1

	F1	F2	F3	F4	F5
BA1	5.79E-13	2.73E-06	4.42E-14	5.56E-15	0
GDEDBA	**9.87E-121**	**4.27E-61**	**0**	**0**	**0**

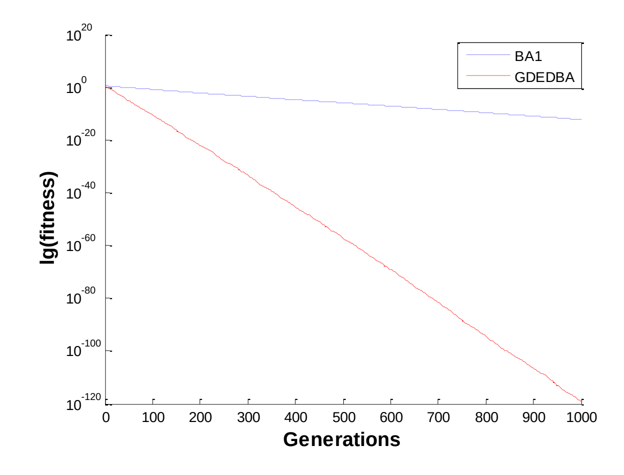


图 2　GDEDBA 与 BA1 运行 F1 的收敛曲线

Fig. 2　Iterative curve of f1about GDEDBA &BA1

分析表 1 和 2 可得，删除指数递减惯性权重后，算法的收敛速度和收敛精度大幅下降，说明指数递减惯性权重可以有效提高算法的寻优性能。那么，改进后算法的速度更新公式为

$$V_i = wV_i^{t-1} + (X_i - X^*) f_i \quad (8)$$

2.2　高斯扰动过程分析

观察基本蝙蝠算法中局部新解的更新公式(式(4))，可以发现它是由当前全局最优解和随机数与响度的乘积项简单相加构成，新解粒子与当前全局最优粒子之间没有产生信息交流和学习，全靠响度项来调节,这就使得对当前全局最优粒子的有效信息利用能力不高，使新解粒子丧失学习能力，还会带来种群多样性的损失,也不符合生物习性。

因此，这里在产生局部新解的公式 $X_{new}^t = X^{*t} + \varepsilon A^t$ 中，对当前全局最优位置的每一个粒子都加入一个高斯扰动项[23] $Gaussian(\mu, \sigma^2)$，其中：μ 表示均值；σ^2 表示方差；$\mu = 0, \sigma^2 = |X^{*t}_{ij}|$；$|X^{*t}_{ij}|$ 表示第 t 次迭代时种群的全局最优粒子的绝对值。改进后的局部新解产生公式描述如下：

$$X_{new}^t = X^{*t}_{j} + a \oplus Gaussian(\mu, \sigma^2) + \varepsilon A^t \,, \varepsilon \in [-1,1] \quad (9)$$

其中：$\mu = 0, \sigma^2 = |X^{*t}_{ij}|$； $\varepsilon \in [-1,1]$ 是随机数； $A^t = <A_i^t>$ 是所有蝙蝠在这一代里的平均响度； a 表示扰动控制因子；$\oplus$ 表示点对点乘法。

如此,新解的产生是由当前全局最优解、均值为 0，方差

为当前全局最优粒子绝对值的高斯扰动项以及响度项三项构成，更加符合蝙蝠之间会有信息交流和相互学习的生物学特点的实际。

2.3　扰动控制因子

基本蝙蝠算法存在求解不稳定、求解精度不高的缺点。可以观察式(4)，改进之前局部新解的产生实际上仅靠响度项在当前全局最优粒子周围调整产生,具有很大的随机性，往往使新解过大的偏离当前全局最优解，或者调整不足,导致种群多样性的丧失,从而产生求解不稳定和精度不高的缺点。

因此这里设计扰动控制因子 a 来控制高斯扰动项的干扰范围（式（9）），增强算法稳定性和增加种群多样性。关于 a 的取值，本文通过仿真实验来说明，当 a 取值为 1、0.5、0.1、0.05、0.01 时，分别用 GDEDBA 算法对测试函数 F1 和 F2（迭代 1 000 次，维数为 30）分别独立进行求解 30 次取其平均值，结果如表 2 所示。

表 2 a 不同取值下仿真结果对比

Table 2　Simulation results of a under different values

	a =1	a =0.5	a =0.1	a =0.05	a =0.01
F1	3.68E-24	5.08E-62	**3.19E-117**	7.50E-104	3.85E-50
F2	5.19E-16	1.06E-34	**4.62E-60**	2.824E-52	4.95E-26

通过观察可以发现，在 a =0.1 时函数的求解结果最好，所以，本文中 a 取值为 0.1，可以保证算法的最优性能。

经过以上分析，高斯扰动项能有效利用当前全局最优位置的信息来保证新解中所有粒子之间的信息交流与学习，一方面可以有效帮助粒子避免陷入局部最优，提高算法搜索精度，另一方面提高了种群多样性，加快了算法收敛速度，可以有效提高算法的寻优性能。

2.4　算法实现流程

a)设置相关参数并初始化蝙蝠种群，随机生成蝙蝠种群的脉冲频率 f_i、位置 X_i、速度 V_i、脉冲速率 r_i 和声音响度 A_i；

b)计算初始化种群的函数适应度值，并判断是否达到迭代终止条件；

c)按照式（7）计算指数递减策略因子 w，用式（1）（8）（3）调整蝙蝠个体的频率并更新速度和位置，找到当前最优位置 X^{*t} 并记录下来；

d)生成一个随机数 rand，并判断是否(rand> r_i)，利用式（9）生成高斯扰动项，在当前迭代次数的全局最优位置附近生成新解 X^t_{new}；

e)判断是否满足随机数 rand< A_i 且适应度值 $f(X^t_{new}) < f(X^{*t})$，同时满足就接受这个新的解，此时 X^t_{new} 成为当前全局最优解，并计算出 X^t_{new} 对应的的适应度值 f_{NEW}，然后按照式（5），（6）更新 r_i 和 A_i，本文将 r_i 和 A_i 设置为一个固定的常数；

f)判断 f_{NEW} 是否为最优适应度值，找到当前全局最优解 X^*，并记录此时的适应度值为目标函数的最优解；

g)判断是否满足算法结束条件，若满足，转到步骤 h)，若未满足，返回步骤 c)；

h)显示全局最优解和最优目标函数适应度值。

为了更加直观地理解改进蝙蝠算法的实现流程，改进蝙蝠算法的算法实现流程如图 3 所示。

3　仿真实验

3.1　算法测试函数

为了测试改进算法的寻优性能，选取 15 个经典测试函数[3,24,25]，通过对基本蝙蝠算法（BA）[1]、标准粒子群算法（PSO）[26]、基本布谷鸟算法（CS）[27]做横向对比，通过对改进蝙蝠算法指数递减权重蝙蝠算法（IWBA）[28]、动态调整惯性权重的自适应蝙蝠算法（DAWBA）[29]作纵向对比，分别进行寻优计算并记录结果。测试函数表 3 所示。

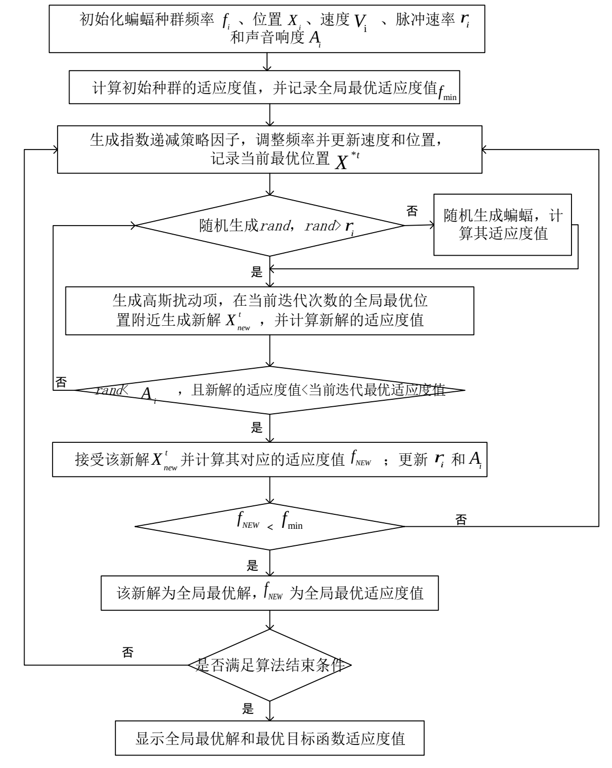


图 3　改进蝙蝠算法的算法流程

Fig. 3　Flow chart of improved bat algorithm

表 3 列出了 15 个测试函数的函数名、对应的函数表达式以及理论最优值。为了方便下文描述，对其进行编号。函数 F1 是非线性连续单峰函数，难以求到最优值，常用于检验算法的求解精度高低；函数 F2 是连续的平滑多峰函数，当自变量趋近于无穷大时，函数会形成大量局部极值区域，具有较高的寻优难度；函数 F3 由于其各维上的变量是密切相关、互相作用的，函数解释变量随着自量变而改变，函数的分布中存在大量局部极值因此很难取得最优解，可以检测算法跳出局部的能力；函数 F4 有 1 个全局极小值点 f(0,0)=0，该函数在狭长的全局极值点周围拥有很多的局部极值，主要用以测试算法的收敛性能；函数 F5 是一个复杂的多峰问题，具有大量的局部极值点，主要用于检验算法的群体多样性；函数 F6 是虽然单峰连续函数，具有一个全局极小点 f(1,1)=0，但它却是病态的（螺旋型），它在全局极小点邻近的狭长区域内取值变化极为缓慢，会发生振荡，可用于评价算法的收敛性能；函数 F7~F10 以及 F14，最小值均为非零常数，其中函数 F7 共有 760 个局部极小值，F9 有 6 个局部极小点，常用来测试算法的求解成功率；函数 F11~函数 13 以及函数 15 用来测试算法的求解稳定性和求解成功率。

通过上述分析可知，文中选取多种不同类型的测试函数，以求全方位测试算法性能，其中不乏求解难度极高的测试函数，这里为了增加求解难度，把所有函数的求解维度都设为 30 维。

3.2　实验环境及参数设置

在仿真实验中，仿真环境的硬件配置为 Windows7 旗舰版操作系统，CPU 为 Intel(R) Core i5 2.40 GHz 处理器，6 GB 内存，仿真环境的软件配置为 MATLABR2014a。

为保证比较的公平性，这里对各个算法的参数作出说明。对于 BA、IWBA、DAWBA、GDEDBA 的参数设置都设置为

种群规模 n=40，响度 A=0.25，脉冲 r =0.5，最大频率 f_{max} =2，最小频率 f_{min} =0；对于 PSO，参数设置为种群规模 n=40，学习因子 c1=2，c2=2，惯性权重 w=0.5；对于 CS，参数设置为种群规模 n=40，概率 pa=0.25；以上所有算法迭代次数都设为 1 000 次；为了提高求解难度，证明算法的优越性，15 个测试函数的求解维数都设为 30 维。

表 3　测试函数

Table 3　Test function

编号	函数名	函数表达式	最优值
F1	Sphere	$f_1(x)=\sum_{i=1}^{n} x_i^2$	0
F2	Schwegel's Problem 2.22	$f_2(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	0
F3	Griewank	$f_3(x)=\frac{1}{4000}\sum_{i=1}^{n} x_i-\prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}})+1$	0
F4	Ackley's	$f_4(x)=20+e-20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2})-\exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))$	0
F5	Rastrigin's	$f_5(x)=\sum_{i=1}^{n}[x_i^2-10\cos(2\pi x_i)+10]$	0
F6	Rosenbrock's	$f_6(x)=\sum_{i=1}^{n-1}[100(\mathrm{x}_{i+1}-\mathrm{x}_i{}^2)^2+(1-\mathrm{x}_i{}^2)]$	0
F7	Hansen	$f_7(x)=\sum_{i=1}^{n} i\cdot\cos((i-1)\cdot x_1+i)\cdot\sum_{j=1}^{n} j\cdot\cos((j+1)\cdot x_2+j)$	-176.54
F8	Branin	$f_8(x)=(x_2-\frac{5.1}{4\pi}x_1^2-6)^2+10\cdot(1-\frac{1}{8\pi})\cos x_1+10$	0.39789
F9	Six-Hump Camel-Back	$f_9(x)=4x_1^2-2.1x_1^4+\frac{x_1^6}{3}+x_1x_2-4x_2^2+4x_2^4$	-1.0316
F10	Goldstein-Price	$f_{10}(x)=[1+(x_1+x_2+1)^2\cdot(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]$ $\times[30+(2x_1-3x_2^2)^2\cdot(18-32x_1+12x_1^2+48x_2^2-36x_1x_2+27x_2^2)]$	3
F11	Tabel	$f_{11}=10^6 x_i+\sum_{i=2}^{n} x_i^2$	0
F12	Matyas	$f_{12}=0.26(x^2+\mathrm{y}^2)-0.48xy$	0
F13	Eggcrate	$f_{13}=x^2+y^2+25(\sin^2 x+\sin^2 y)$	0
F14	Easom	$f_{14}(x)=-\cos(x_1)\cos(x_2)\exp[-((x_1-\pi)^2+(x_2-\pi)^2)]$	-1
F15	Axis parallel hyper-ellipsoid	$f_{15}(x)=\sum_{i=1}^{d}\sum_{j=1}^{i} x_j^2$	0

3.3　寻优性能分析

为了减少随机因素对算法求解结果的影响，把每个函数分别独立运行 50 次，分别计算 15 个测试函数独立运行 50 次的最优解（Best）、平均值（Avg）和方差（Var），并把求解精度最高的解进行加粗，结果对比如表 4 所示。

由表 4 可以看出，对于 15 个测试函数而言，GDEDBA 可以对其中的 11 个求到最优解，其余 4 个的寻优结果远远优于其他算法，说明改进算法的求解达优率很高。仔细分析，对于函数 F7~F10 和 F14，本文选取的五个算法以及 GDEDBA 都可以求到最优解，说明本文选取的所有算法在求解函数 F7~F10 和 F14 都显示了优良的求解性能，但是对于函数 F7 和 F10 来说，GDEDBA 的求解结果方差最小，说明改进后的算法稳定性更强；对于函数 F4~F6，PSO，IWBA 和 GDEDBA 可以求到最优解，而其余算法求不到最优解；对于函数 F1,F2,F11 和 F15 本文选取的五个算法以及改进后的算法都没有求到最优解，但是 GDEDBA 的求解精度高出其余算法近百倍的数量级，而且求解方差远远高于其余算法，说明 GDEDBA 的稳定性更强；而对于函数 F3,F12 和 F13，只有本文的改进算法 GDEDBA 可以求到最优解，其余算法都没有求到最优解。经过以上分析可以得出，本文设计的改进算法在在寻优性能上明显优于其他算法，显示了更高的寻优精度和更高的稳定性。

3.4　收敛性能分析

算法的收敛速度是衡量算法性能的重要指标，算法的收敛图可以更加直观地对比各种算法的收敛性能。图 4~18 展示了 BA、PSO、CS、IWBA、DAWBA 和 GDEDBA 这 6 种优化算法在求解 15 个标准测试函数时的适应度值收敛曲线。图中横轴为迭代次数，纵轴为适应度值。另外，为了更清楚地显示收敛情况，把函数 F1~F6，F11~F13 和 F15 的纵轴取对数。

由图4~ 8、图14~ 16 以及图18 可知，在求解函数 F1~F5、F11~F13 以及 F15 时，GDEDBA 的收敛曲线向图的右下角迅速下降，尤其是图 6~8 对应的函数 F3~F5，GDEDBA 的收敛曲线几乎接近垂直下降，收敛非常快，远远高于其余算法；观察图 6~8 可发现，其余 5 种算法在收敛过程中出现了不同程度的阶梯状，观察图 14~16 发现，PSO 在求解函数 F11~F13 时，在迭代前期出现了剧烈震荡，而 GDEDBA 收敛曲线平滑，由收敛曲线可以看出，GDEDBA 求解稳定性很强，显示了很强的鲁棒性。由图 10~13 以及图 17 可知，求解函数 F7~F10 以及 F14 时，GDEDBA 的收敛曲线几乎与其余算法重合，并且由上文可知 F7~F10 以及 F14 都可以求到最优解，两者互为证明，算法的收敛速度与收敛精度是存在一定联系的。由图 9 可知，PSO 的收敛速度快于其余算法，而 DGEDBA 和 IWBA 的收敛曲线呈现重合趋势，而由上文得知，对于函数 F6，PSO、IWBA 和 GDEDBA 都可以求到最优解。

表 4　寻优精度的对比结果

Table 4　Comparison results of optimization accuracy

函数	算法	Best	Avg	Var
F1	BA	9.0151E-04	1.4130E-03	3.4986E-08
	PSO	2.0146E-01	5.3673E-01	3.5194E-02
	CS	1.6948E-05	5.1693E-05	7.2214E-10
	IWBA	6.0092E-05	2.8104E-04	8.0002E-09
	DAWBA	3.7525E-07	1.1544E-06	3.4034E-13
	GDEDBA	**4.4294E-123**	**2.1932E-117**	**5.1902E-233**
F2	BA	1.2693E-01	2.1723E-01	1.78E-02
	PSO	1.309	2.680037255	0.425097685
	CS	1.2434E-01	2.6981E-01	6.1909E-03
	IWBA	7.5163E-02	3.2902E-01	6.10E-02
	DAWBA	3.4754E-03	1.3185E-01	3.92E-02
	GDEDBA	**3.9700E-63**	**3.4265E-60**	**5.8851E-119**
F3	BA	3.6417E-05	7.2499E-05	1.13885E-10
	PSO	1.0912E-02	2.2616E-02	6.3948E-05
	CS	5.1171E-05	7.0376E-04	1.2800E-06
	IWBA	8.7483E-06	3.0170E-05	1.19875E-10
	DAWBA	1.9098E-08	8.2571E-08	2.42595E-15
	GDEDBA	**0**	**0**	**0**
F4	BA	4.9913E-05	2.9146E-04	1.64453E-08
	PSO	0	0	0
	CS	0	0	0
	IWBA	0	0	0
	DAWBA	9.7607E-08	2.0547E-06	1.96856E-12
	GDEDBA	**0**	**0**	**0**
F5	BA	1.78E-15	1.31E-10	4.97312E-20
	PSO	0	0	0
	CS	0	0	0
	IWBA	0	0	0
	DAWBA	0	6.77E-14	1.7116E-26
	GDEDBA	**0**	**0**	**0**
F6	BA	1.49E-09	8.96E-08	8.48199E-15
	PSO	0	0	0
	CS	0	1.1434E-25	1.4415E-49
	IWBA	0	0	0
	DAWBA	5.25E-13	9.11E-11	2.54143E-20
	GDEDBA	**0**	**0**	**0**
F7	BA	-176.54	-171.1060784	176.2489323
	PSO	-176.54	-172.9331373	135.213922
	CS	-176.54	-176.36676	85.762971
	IWBA	-176.54	-171.7623529	196.5068224
	DAWBA	-176.54	-174.7129412	54.47850918
	GDEDBA	**-176.54**	**-175.3219608**	**37.07565208**
F8	BA	0.39789	0.39789	5.02899E-32
	PSO	0.39789	0.39789	5.02899E-32
	CS	0.39789	0.39789	2.87239E-32
	IWBA	0.39789	0.39789	5.02899E-32
	DAWBA	0.39789	0.39789	5.02899E-32
	GDEDBA	**0.39789**	**0.39789**	**5.02899E-32**
F9	BA	-1.0316	-1.0316	4.52609E-31
	PSO	-1.0316	-1.0316	4.52609E-31
	CS	-1.0316	-1.0316	4.59582E-31
	IWBA	-1.0316	-1.0316	4.52609E-31
	DAWBA	-1.0316	-1.0316	4.52609E-31
	GDEDBA	**-1.0316**	**-1.0316**	**4.52609E-31**
F10	BA	3	4.058823529	28.01647059
	PSO	3	3	0
	CS	3	3	0
	IWBA	3	4.588235294	41.16705882
	DAWBA	3	4.058823529	28.01647059
	GDEDBA	**3**	**3**	**0**

续表 4

函数	算法	Best	Avg	Var
F11	BA	3.5727E-04	6.9015E-01	0.477236782
	PSO	1.8816E-123	2.6125E-110	1.9336E-218
	CS	2.1420E-25	9.4176E-22	5.7827E-42
	IWBA	1.1134E-51	7.9783E-46	5.51478E-90
	DAWBA	3.5727E-04	6.9015E-01	0.477236782
	GDEDBA	**1.9677E-270**	**6.8276E-258**	**0**
F12	BA	1.3617E-11	8.8821E-10	6.2287E-19
	PSO	7.0720E-161	3.1377E-148	2.1632E-294
	CS	1.3575E-45	1.6298E-40	5.3858E-79
	IWBA	4.7641E-76	2.9422E-69	3.1479E-136
	DAWBA	1.3617E-11	8.8821E-10	6.2287E-19
	GDEDBA	**0**	**0**	**0**
F13	BA	2.7643E-09	2.3366E-07	5.09804E-14
	PSO	2.3539E-158	1.5851E-147	1.2409E-292
	CS	1.1081E-39	1.7153E-34	5.3844E-67
	IWBA	1.6051E-74	4.9126E-67	1.053E-131
	DAWBA	2.7643E-09	2.3366E-07	5.09804E-14
	GDEDBA	**0**	**0**	**0**
F14	BA	-1	-1	0
	PSO	-1	-1	0
	CS	-1	-1	0
	IWBA	-1	-1	0
	DAWBA	-1	-1	0
	GDEDBA	**-1**	**-1**	**0**
F15	BA	4.1359E-02	2.4627E-02	3.5267E-05
	PSO	3.0380E+00	6.6265E+00	3.5508E+00
	CS	2.0909E-04	5.9789E-04	8.0044E-08
	IWBA	5.5066E-03	2.1726E-02	1.5174E-04
	DAWBA	6.7472E-06	2.3829E-05	1.3612E-10
	GDEDBA	**1.1240E-121**	**6.3033E-117**	**2.7792E-232**

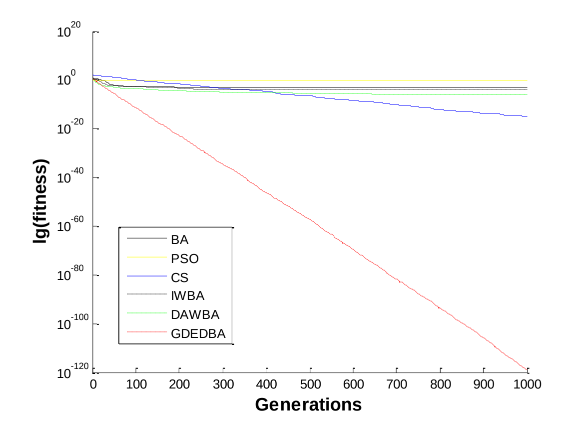


图 4　函数 F1 的收敛曲线

Fig. 4　Convergence curves of function F1

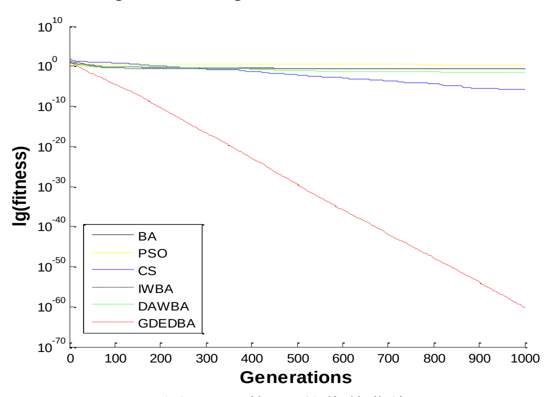


图 5　函数 F2 的收敛曲线

Fig. 5　Convergence curves of function F2

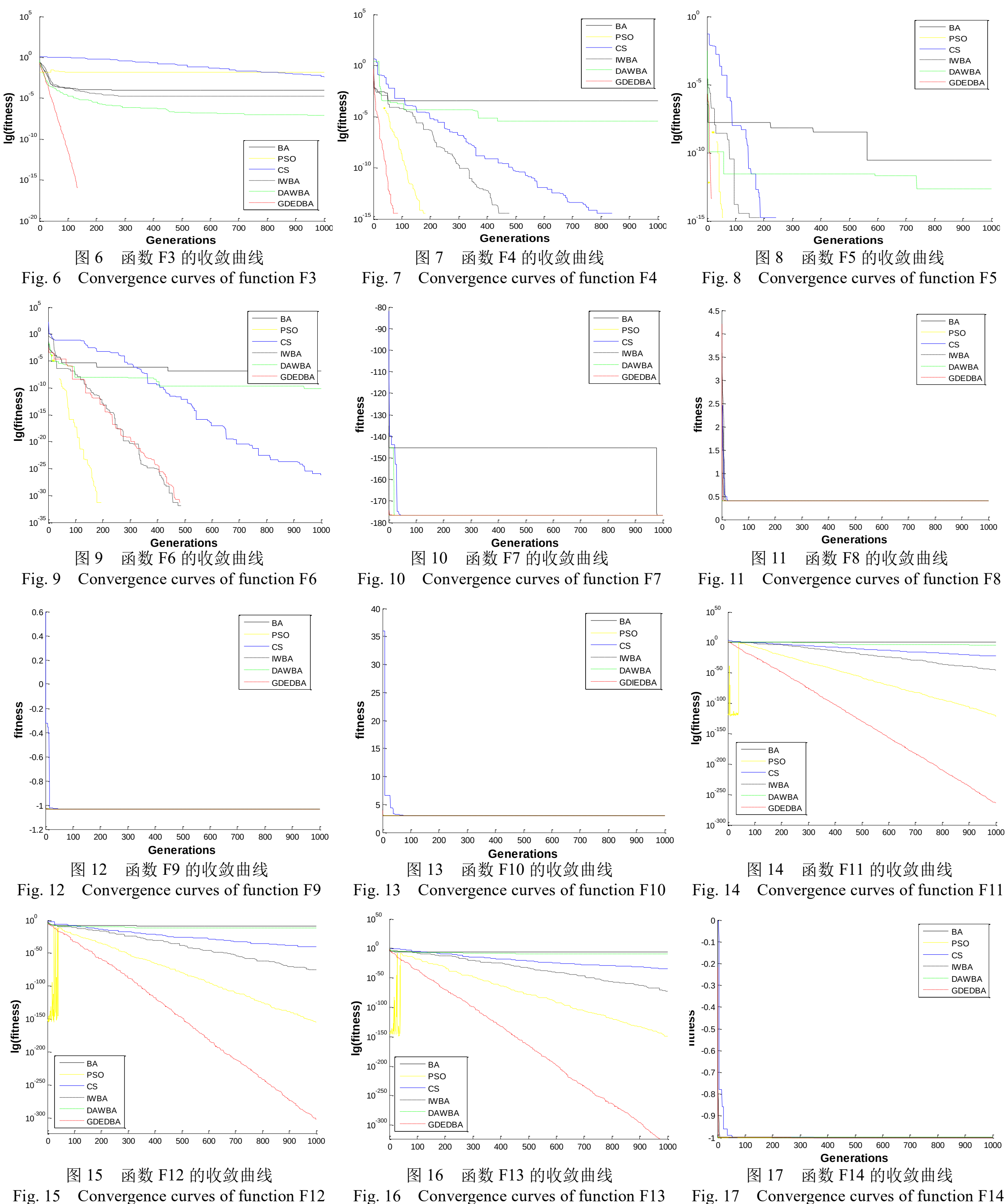


图 6　函数 F3 的收敛曲线

Fig. 6　Convergence curves of function F3

图 7　函数 F4 的收敛曲线

Fig. 7　Convergence curves of function F4

图 8　函数 F5 的收敛曲线

Fig. 8　Convergence curves of function F5

图 9　函数 F6 的收敛曲线

Fig. 9　Convergence curves of function F6

图 10　函数 F7 的收敛曲线

Fig. 10　Convergence curves of function F7

图 11　函数 F8 的收敛曲线

Fig. 11　Convergence curves of function F8

图 12　函数 F9 的收敛曲线

Fig. 12　Convergence curves of function F9

图 13　函数 F10 的收敛曲线

Fig. 13　Convergence curves of function F10

图 14　函数 F11 的收敛曲线

Fig. 14　Convergence curves of function F11

图 15　函数 F12 的收敛曲线

Fig. 15　Convergence curves of function F12

图 16　函数 F13 的收敛曲线

Fig. 16　Convergence curves of function F13

图 17　函数 F14 的收敛曲线

Fig. 17　Convergence curves of function F14

图 18　函数 F15 的收敛曲线

Fig. 18　Convergence curves of function F15

综合以上分析，GDEDBA 在处理单峰多维、多峰多维的问题上可以显示很高的收敛速度与收敛精度，并且稳定性高于其他算法，表明了改进后的算法（GDEDBA）在寻优精度、收敛速度和求解稳定性方面比其余五种算法有显著的优越性，求解性能优良。

4　结束语

针对基本蝙蝠算法后期收敛速度慢、寻优精度低、求解不稳定等不足，通过在产生局部新解的公式中加入均值为 0、方差为当前迭代次数的最优解绝对值的高斯扰动项，加入步长控制因子确保扰动幅度，既提高了算法的精确搜索能力，

又增加种群多样性与活跃性，进一步挖掘蝙蝠算法的仿生性能；对速度更新公式加入指数递减策略使算法迅速进入局部精确搜索，实现对基本蝙蝠算法从全局寻优到局部寻优都做了有效改进来提高算法性能。仿真结果表明，GDEDBA 的寻优精度、收敛速度以及算法稳定性方面均有很大程度的提升，寻优性能优良。

参考文献：

[1] Yang Xinshe. A new meta-heuristic bat-inspired algorithm [C]// Proc of Nature Inspired Cooperative Strategies for Optimization. Berlin Eidelberg: Springer-Verlag, 2010: 65-74.

[2] Tsai P W, Pan J S, Liao B Y, *et al.* Bat algorithm inspired algorithm for solving numerical optimization problems [J]. Applied Mechanics and Materials, 2012, 148-149 (148): 134-137.

[3] 李煜，裴宇航，刘景森. 融合均匀变异与高斯变异的蝙蝠优化算法 [J]. 控制与决策, 2017, 32 (10): 1775-1781. (Li Yu, Pei Yuhang, Liu Jingsen. Bat optimal algorithm combined uniform mutation with Gaussian mutation [J]. Control and Decision, 2017, 32 (10): 1775-1781.)

[4] Yang Xinshe. Bat algorithm for multi-objective optimization [J]. International Journal of Bio-Inspired Computation, 2011, 3 (5): 267-274.

[5] Kang M, Kim J, Kim J M, *et al.* Reliable fault diagnosis for incipient low-speed bearings using fault feature analysis based on a binary bat algorithm [J]. Information Sciences, 2015, 294 (2): 423-438.

[6] Abd-Elazim S M, Ali E S. Load frequency controller design via BAT algorithm for nonlinear interconnected power system [J]. Electrical Power and Energy Systems, 2016, 77 (5): 166-177.

[7] Singh K, Vasant P. Elamvazuthi I, *et al.* PID tuning of servo motor using bat algorithm [J]. Procedia Computer Science, 2015, 60 (1): 1798-1808.

[8] 马邦雄，叶春明. 基于蝙蝠退火算法的无等待流水线调度问题研究 [J]. 数学理论与应用, 2014, 34 (1): 92-101. (Ma Bangxiong, Ye Chunming. A bat annealing algorithm for solving the no-wait flow shop scheduling problem [J]. Mathematical Theory and Applications, 2014, 34 (1): 92-101.)

[9] 金伟健，王春枝. 基于蝙蝠算法的云计算资源分配研究 [J]. 计算机应用研究, 2015, 32 (4): 1184-1187. (Jin Weijian, Wang Chunzhi. Study on bat algorithm in cloud computing resources allocation [J]. Application Research of Computers, 2015, 32 (4): 1184-1187.)

[10] Ye Zhiwei, Wang Mingwei, Liu Wei, *et al.* Fuzzy entropy based optimal thresholding using bat algorithm [J]. Applied Soft Computing, 2015, 31 (6): 381-395.

[11] 陈海挺. 改进蝙蝠算法优化极限学习机的图像分类 [J]．激光杂志, 2014, 35 (11): 26-29. (Chen Haiting. Image classification based on extreme learning machine optimized by improved bat algorithm [J]. Laser Journal, 2014, 35 (11): 26-29.)

[12] Selim Y, Ecir U. Kücüksille. A new modification approach on bat algorithm for solving optimization problems [J]. Applied Soft Computing, 2015, 28 (12): 259-275.

[13] Meng Xianbing, Liu Yu, Zhang Hengzhen, *et al.* A novel bat algorithm with habitat selection and doppler effect in echoes for optimization [J]. Expert Systems with Applications, 2015, 42 (17): 6350-6364.

[14] Luo Jun, Liu Liheng, Wu Xianyi. A double-subpopulation variant of the bat algorithm [J]. Applied Mathematics and Computation, 2015, 263 (7): 361-377.

[15] Nakamura R Y M, Pereira L, Costa K, *et al.* BBA: a binary bat algorithm for feature selection [C]// Proc of the 25th SIBGRAPI Conference on Graphics, Patterns and Images. Piscataway: IEEE Press, 2012: 291-297.

[16] He Xingshi, Ding Wenjing, Yang Xinshe. Bat algorithm based on simulated annealing and Gaussian perturbations [J]. Neural Computing and Applications, 2013, 25 (2): 459-468.

[17] 戚远航，蔡延光，蔡颢，等. 旅行商问题的混沌混合离散蝙蝠算法 [J]. 电子学报, 2016, 44 (10): 2543-2547. (Qi Yuanhang, Cai Yanguang, Cai Hao, *et al.* Chaotic hybrid discrete bat algorithm for traveling salesman problem [J]. Acta Electronica Sinica, 2016, 44 (10): 2543-2547.)

[18] Premkumar K, Manikandan B V, Speed control of Brushless DC motor using bat algorithm optimized adaptive neuro-fuzzy inference system [J]. Applied Soft Computing, 2015, 32 (7): 403–419.

[19] Shi Yuhui, Eberhart R C. Empirical study of particle swarm optimization [C]// Proc of Congress on Evolutionary Computation. Washington DC: IEEE Press, 1999: 1945-1949.

[20] Shi Yuhui, Eberhart R C. Fuzzy adaptive particle swarm optimization [C]// Proc of Congress on Evolutionary Computation. Piscataway,NJ: IEEE Press, 2001: 101-106.

[21] Eberhart R C, Shi Yuhui. Tracking and optimizing dynamic systems with particle swarms [C]// Proc of Congress on Evolutionary Computation. Piscataway,NJ: IEEE Press, 2001: 101-106.

[22] 陈贵敏，贾建援，韩琪. 粒子群优化算法的惯性权值递减策略研究 [J]. 西安交通大学学报, 2006, 40 (1): 53-61. (Chen Guimin, Jia Jianyuan, Han Qi. Study on the strategy of decreasing inertia weight in particle swarm optimization algorithm [J]. Journal of Xi’an Jiaotong University, 2006, 40 (1): 53-61.)

[23] 朱德刚，孙辉，赵嘉，等. 基于高斯扰动的粒子群优化算法 [J]. 计算机应用, 2014, 34 (3): 754-759. (Zhu Degang, Sun Hui, Zhao Jia, *et al.* Particle swarm optimization algorithm based on Gaussian disturbance [J]. Journal of Computer Applications, 2014, 34 (3): 754-759.)

[24] Jordehi A R. Chaotic bat swarm optimization (CBSO) [J]. Applied Soft Computing, 2014, 26 (1): 523–530.

[25] 尚俊娜，刘春菊，岳克强，等. 具有自学习能力的变异蝙蝠优化算法及性能仿真 [J]. 系统仿真学报, 2017, 29 (2): 301-307. (Shang Junna, Liu Chunju, Yue Keqiang, *et al.* Variation bat algorithm with self-learning capability and its property analysis [J]. Journal of System Simulation, 2017, 29 (2): 301-307.)

[26] Kennedy J, Eberhart R. Particle swarm optimization [C]// Proc of IEEE International Conference on Neural Networks. Piscataway,NJ: IEEE Press, 1995: 1942-1948.

[27] Yang Xinshe, Deb S. Cuckoo search via Lévy flights [C]// Proc of World Congress on Nature & Biologically Inspired Computing. Piscataway,NJ: IEEE Press, 2009: 210-214.

[28] 宋一民. 基于改进蝙蝠算法的物流中心选址问题研究 [D].开封: 河南大学, 2017. (Song Yimin. Research on logistics center location based on improved bat algorithm [D]. Kaifeng: Henan University, 2017.)

[29] 裴宇航，刘景森，李煜. 一种动态调整惯性权重的自适应蝙蝠算法 [J]. 计算机科学, 2017, 44 (6): 240-244. (Pei Yuhang, Liu Jingsen, Li Yu. An adaptive bat algorithm with dynamically adjusting inertia weight [J]. Computer Science, 2017, 44 (6): 240-244.)